**Introduction to Lucene**

------------

ORGWARE

Contents

Table of Contents

1. Introduction-Definition**3**

1.1 How Lucene works3

1.2 Indexing4

1.3 Queries7

2. Review8

2.1 Network traffic analysis framework for cyber threat detection8

2.2 A Text Mining Approach to Discovering COVID-19 Relevant Factors9

2.3 The Modality of Tourist Advertising Promotion as Place Branding in Greece..10

3. Findout12

3.1 TokenStream12

3.2 Position13

# Introduction – Definition

# Lucene is an open source java full-text search library which makes it easy to add search functionality to an application or website. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

# Lucene is a library that allows the user to index textual data (Word & PDF documents, emails, webpages, tweets etc). It allows you to add search capabilities to your application. There are two main steps that Lucene performs:

# Create an index of documents you want to search.

# Parse query, search index, return results.

### **1.1 How Lucene works**

So, you must be wondering how Lucene can perform very fast full-text searches. Not surprisingly, the answer is that it uses an *index*. Lucene indexes fall into the category of *inverted indexes*. Instead of having a classic index where for every document you have the full list of words (or *terms*) it contains, inverted indexes do it the other way round. For every *term* (word) in the documents, you have a list of all of the documents that contain that term. That is hugely more convenient when performing full text searches.

The reason that inverted indexes work so good can be seen in the following diagrams. Imagine you have 3 very large document. A classic index you be of the form:

Classic index:

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3 | Document1 -> { going, to, dive, into, Apache, Lucene, rich, open, source, full, text, search,... }  Document2 -> { so, must, wonder, Lucene, can, achieve, very, fast, full, text, search, not,... }  Document3 -> { reason, that, inverted, index, work, good ,can, be, seen, following, diagrams,... } | |  |

For every document you have a huge list of all of the terms it contains. In order to find if a document contains a specific term you have to scan, probably sequentially these vast lists.

On the other hand an inverted index would have that form:

Inverted index:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | reason -> { (3,0} }  Lucene -> { (1,6), (2,4) }  full   -> { (1,10),(2,9) }  going  -> { (1,0) }  index  -> { (3,3) }  search -> { (1,11), (2,10)} |

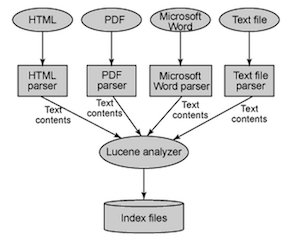
For every term we maintain a list with all of the documents that contain that term, followed by the position of the term inside the document (of course additional information can be kept). Now, when a user searches for the term “Lucene”, we can instantly answer that the term “Lucene” is located inside Document1 in position 6 and inside Document2 in position 4. To sum up, inverted indexes use a very big number of very small lists that can be searched instantly. In contrast a classic index would use a small number of extremely big lists that is impossible to search quickly.

* 1. **Indexing**

Lucene uses an inverted index (mapping of a term to its metadata). This metadata includes information about which files contain this term, number of occurrences etc. The fundamental units of indexing in Lucene are the Document and Field classes:

* A Document is a container that contains one or more Fields.
* A Field stores the terms we want to index and search on. It stores a mapping of a key (name of the field) and a value (value of the field that we find in the content).

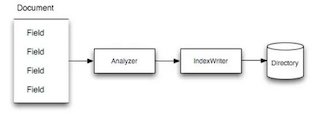
Here is a diagram describing the steps Lucene takes when indexing content (Source: Lucene in Action, Figure 2.1).



In order to be able to index documents of various types, Lucene needs to be able to extract the test from the given document into a format that it can parse. Apache Tika is one framework that parses documents and extracts text content.

**Analyze**

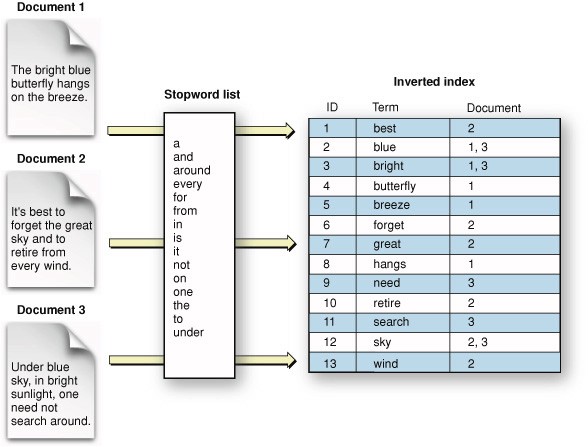
This process filters and cleans up the text data. The text data goes through several steps (for example: extracting words, removing common (stop) words, make words lowercase etc) and converts the text into tokens that can be added to the index. The picture to the right shows the indexing process which results in an inverted index being stored on the underlying filesystem. See below for an example of an inverted index.



**Write Index**

Lucene uses an inverted index data structure for storing the Fields we want to search on. An inverted index uses the tokens as the lookup key to find the documents which contains that token. It maps the content to its location. The index can be physically stored as part of a Directory (either in a file system or in memory).

Below is an example of an inverted index. Logically, this represents the result of the indexing process.

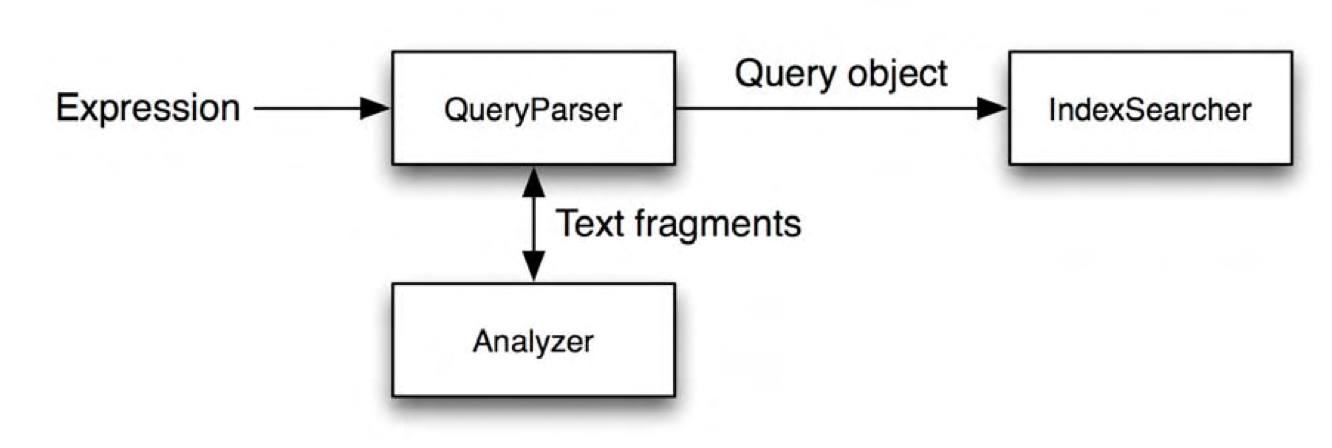


# **Searching**

Once our documents are indexed, we will need to add search functionality. All queries of the index are done through the IndexSearcher. Given a search expression, we parse the query, create a QueryParser and search the index for results. The results are returned as TopDocs which contain ScoreDocs, which contain the document IDs and the confidence scores of the results that match the query. The fundamental classes for searching are:

* IndexSearcher — Provides “read-only” access to the index. Exposes several search methods that take in a Query object and return the top n “best” TopDocs as the result. This class is the counter part to the IndexWriter class used for creating/updating indexes.
* Term — Basic unit for searching. Counter part to the Field object used in indexing. We create a certain Field when indexing (for ex: “Name” : “Chuck Norris”) and we use Terms in a TermQuery when searching. It contains the same mapping from the name of the field to the value
* Query: Lucene provides several types of Queries, including TermQuery, BooleanQuery, PrefixQuery, WildcardQuery, PhraseQuery, and FuzzyQuery. Each type of query provides a unique way of searching the index.
* QueryParser: Parses a human-readable query (for ex: “opower AND arlington”) into Query object that can be used for searching.
* TopDocs — Container for pointers to N search results. Each TopDoc contains a document ID and a confidence score.

Here is an overview of the process described above:



**1.3 Queries.**

* fuzzyQuery:- FuzzyQuery is used to search documents using fuzzy implementation that is an approximate search based on the edit distance algorithm.
* [TermQuery](https://www.tutorialspoint.com/lucene/lucene_termquery1.htm) :-This class acts as a core component which creates/updates indexes during the indexing process.
* [PhraseQuery](https://www.tutorialspoint.com/lucene/lucene_phrasequery.htm) :-Phrase query is used to search documents which contain a particular sequence of terms.

# 2.Review

2.1 Network traffic analysis framework for cyber threat detection

**Abstract:**

The growing sophistication of attacks and newly emerging cyber threats requires advanced cyber threat detection systems. Although there are several cyber threat detection tools in use, cyber threats and data breaches continue to rise. This research is intended to improve the cyber threat detection approach by developing a cyber threat detection framework using two complementary technologies, search engine and machine learning, combining artificial intelligence and classical technologies. In this design science research, several artifacts such as a custom search engine library, a machine learning-based engine and different algorithms have been developed to build a new cyber threat detection framework based on self-learning search and machine learning engines. Apache Lucene.Net search engine library was customized in order to function as a cyber threat detector, and Microsoft ML.NET was used to work with and train the customized search engine. This research proves that a custom search engine can function as a cyber threat detection system. Using both search and machine learning engines in the newly developed framework provides improved cyber threat detection capabilities such as self-learning and predicting attack details. When the two engines run together, the search engine is continuously trained by the machine learning engine and grow smarter to predict yet unknown threats with greater accuracy. While customizing the search engine to function as a cyber threat detector, this research also identified and proved the best algorithms for the search engine based cyber threat detection model. For example, the best scoring algorithm was found to be the Manhattan distance. The validation case study also shows that not every network traffic feature makes an equal contribution to determine the status of the traffic, and thus the variable-dimension Vector Space Model (VSM) achieves better detection accuracy than ndimensional VSM. Although the use of different technologies and approaches improved detection results, this research is primarily focused on developing techniques rather than building a complete threat detection system. Additional components such as those that can track and investigate the impact of network traffic on the destination devices make the newly developed framework robust enough to build a comprehensive cyber threat detection appliance.

## **Conclusion:**

This chapter concludes with the overall outcome of the research, a list of contributions, limitations of the research, and recommendations for future research. This research shows the possibility of a search engine serving as a cyber threat detection framework. It also shows that the search engine-based threat detection engine can be reinforced by a machine learning-based engine. As a result, using the search- and machine learning-based engines working together improves the capability of the cyber threat detection system. Using two different technologies, search engines and machine learning, the newly developed framework was focused on the analysis of network traffic for cyber threat detection. Several artifacts such as traffic feature processors, detection engines, an alert service, and dashboard applications have been developed. A search engine library, Lucene was customized to function as a threat detection tool. During customization, several algorithms and techniques have also been investigated. The developed framework was tested and validated. In this chapter we will explore the outcomes of the research, its contributions and limitations, and recommendations for future research.

Link: <https://core.ac.uk/download/pdf/327163105.pdf>

2.2 A Text Mining Approach to Discovering COVID-19 Relevant Factors

**Abstract:**

This paper describes a text mining approach that utilises the PyLucene search engine and the GrapeNLP grammar engine for extracting links between temperature, humidity and the spread of COVID-19, from a vast collection of scientific publications. The approach was developed in response to a Kaggle challenge from a consortium of research groups to develop text and data mining techniques that can assist the medical community in finding answers to a series of important questions on COVID-19. For this challenge, a large corpus of scientific publications known as the COVID-19 Open Research Dataset (CORD-19) was provided by the consortium. The approach presented in this paper was winner of the competition task of extracting key insights and building summary tables of COVID-19 relevant factors such as temperature and humidity.

## **Conclusion**

This paper described a new approach to uncovering key insights and discoveries on COVID-19 from a large corpus of scientific publications known as [1] the COVID-19 Open Research Dataset (CORD-19). It was developed in response to a call for new techniques that can assist the medical community in finding answers to a series of important scientific questions on COVID-19. Providing automated answers to these questions can help clinical experts to rapidly validate the results of recent research and propose new experimental hypotheses to help manage and reduce the impact of the COVID-19 pandemic.

The approach is based on two open-source components: the Apache Lucene search engine for keyword search, and the GrapeNLP grammar engine for further refinement of the matches found, enabling search functionality for more complex linguistic structures.

The approach was also validated by the CORD-19 Kaggle competition jury panel, who selected it as the best for creating summary tables on COVID-19 relevant factors. The approach was demonstrated using the effect of temperature and humidity on transmission as a use case. Grammars can be further refined via the Unitex graphical user interface, which is also open source, and extended to a wider range of questions. Indeed, the approach was also demonstrated on identifying relevant COVID-19 risk factors 10.

While deep learning approaches have also gained traction for addressing complex information retrieval tasks, they have short comings, such as the reliance on labelled data and the need for long training times and expensive hardware. They also suffer from lack of explainability and interpretability. The approach presented in this paper is efficient and has added benefit of clearer explainability in terms of the retrieval mechanism, as the grammar can easily be applied and reviewed by non-specialists. This COVID-19 pipeline is open sourced and is freely available for users, researchers and developers that may require to search papers using potentially complex linguistic structures.

Link: <https://ieeexplore.ieee.org/abstract/document/9313149>

2.3 The Modality of Tourist Advertising Promotion as Place Branding in Greece. A Visual Retrieval Framework

**Abstract**

Print advertisements are a good ground for studying phenomena of textual and visual diversity. As a combination of text and image, the advertisements represent a large number of varieties of social and commercial reality through which they reproduce and reinforce specific ideologies and correspondingly imaginative ones. Advertising is a reason for the commodity society and the mass culture that it entails, and as such defines textual (linguistic) behavior in the sense that it aligns the projected with linguistic and visual homogeneity and consequently stereotypes resulting from this. The aim of this article is to record and visualize the textual and visual signs in the printed advertisements concerning the way of showing Greece as a tourist destination between 1920 and 2019. The theoretical framework of this reading is based on two theoretical frameworks: first, in polygraphism and secondly in that of destination marketing and place branding. Furthermore, the technical framework provides a simple way to retrieve images and photos based on color and texture characteristics of the presented type of advertisements. Then, the referred framework creates a Lucene index of image features for content-based image retrieval using local and global state-of-the-art methods. The ultimate goal of this recording is to reveal and retrieve the textual and visual varieties (mainly of the latter because of their unequal frequency in the empirical material), which indicates the self-determination of the Greeks, in relation to history and ethnography.

## **Conclusion**

As we have seen from the analysis of the content of the ads that make up the experimental material of the study, Throughout this period of impact, we have noted with interest the shifting of cleaning products by making interesting conclusions about advertising and marketing as well as using the Google Cloud AutoML platform to create an end-to-end advertising classification model .

Link:<https://www.researchgate.net/profile/Anargyros-Koumparelis/publication/347463273_The_Modality_of_Tourist_Advertising_Promotion_as_Place_Branding_in_Greece_A_Visual_Retrieval_Framework/links/5fdcc638a6fdccdcb8ddf9ae/The-Modality-of-Tourist-Advertising-Promotion-as-Place-Branding-in-Greece-A-Visual-Retrieval-Framework.pdf>

# Find Out

3.1 How we can use TokenStream? (include code snippet)

TokenStream is an output of the analysis process and it comprises of a series of tokens. It is an abstract class.

public abstract class TokenStream

extends AttributeSource

implements Closeable

A TokenStream enumerates the sequence of tokens, either from [Field](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/document/Field.html)s of a [Document](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/document/Document.html) or from query text.

This is an abstract class; concrete subclasses are:

* [Tokenizer](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/Tokenizer.html), a TokenStream whose input is a Reader; and
* [TokenFilter](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/TokenFilter.html), a TokenStream whose input is another TokenStream.

A new TokenStream API has been introduced with Lucene 2.9. This API has moved from being [Token](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/Token.html)-based to [Attribute](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/Attribute.html)-based. While [Token](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/Token.html) still exists in 2.9 as a convenience class, the preferred way to store the information of a [Token](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/Token.html) is to use [AttributeImpl](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/AttributeImpl.html)s.

TokenStream now extends [AttributeSource](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/AttributeSource.html), which provides access to all of the token [Attribute](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/Attribute.html)s for the TokenStream. Note that only one instance per [AttributeImpl](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/AttributeImpl.html) is created and reused for every token. This approach reduces object creation and allows local caching of references to the [AttributeImpl](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/util/AttributeImpl.html)s. See [incrementToken()](https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/analysis/TokenStream.html#incrementToken()) for further details.

**Class constructors**

* **protected TokenStream() :** A TokenStream that uses the default attribute factory.
* **protected TokenStream(AttributeSource.AttributeFactory factory):** A TokenStream that uses the supplied AttributeFactory for creating new Attribute instances.
* **protected TokenStream(AttributeSource input) :** A TokenStream that uses the same attributes as the supplied one.

3.2 What is the use of position?

The use of position is list the ordinal positions of each occurrence of a term in a document.

# GIT

Add this prepared content to the same repo :

https://gitlab.com/Dipendra123/maven-webscrape-project